



US005761723A

**United States Patent** [19]

Black et al.

[11] Patent Number: **5,761,723**[45] Date of Patent: **Jun. 2, 1998****[54] DATA PROCESSOR WITH BRANCH PREDICTION AND METHOD OF OPERATION**

[75] Inventors: **Bryan P. Black; Marvin Denman; Mark A. Kearney; Seungyoon Peter Song**, all of Austin, Tex.

[73] Assignee: **Motorola, Inc.**, Schaumburg, Ill.

[21] Appl. No.: **637,189**

[22] Filed: **Apr. 8, 1996**

**Related U.S. Application Data**

[63] Continuation of Ser. No. 192,292, Feb. 4, 1994, abandoned.

[51] Int. Cl.<sup>6</sup> ..... **G06F 12/08**

[52] U.S. Cl. .... **711/144; 711/145; 395/586**

[58] Field of Search ..... 395/584, 585, 395/586, 421.03, 464, 462; 711/125, 128, 137, 144, 204, 211, 213, 145

**[56] References Cited****U.S. PATENT DOCUMENTS**

4,654,819	3/1987	Stuffer et al.	395/489
5,093,778	3/1992	Favor et al.	395/587
5,136,697	8/1992	Johnson	395/586
5,163,140	11/1992	Stiles et al.	711/140
5,226,130	7/1993	Favor et al.	395/585
5,265,213	11/1993	Werser et al.	395/587
5,319,760	6/1994	Mason et al.	395/416
5,327,536	7/1994	Suzuki	395/585
5,327,547	7/1994	Stiles et al.	711/137

5,353,421	10/1994	Emma et al.	395/587
5,394,529	2/1995	Brown, III et al.	395/587
5,404,467	4/1995	Saba et al.	395/383
5,414,822	5/1995	Saito et al.	395/587
5,423,011	6/1995	Blaner et al.	395/587
5,423,048	6/1995	Jager	395/383

**OTHER PUBLICATIONS**

Intel Corporation, "Pentium Family User's Manual", vol. 3: Architecture and Programming Manual, pp. 16-1 through 16-31 (1994).

Advanced Micro Devices, Inc. "Am29000 and Am29005 User's Manual and Data Sheet 32-Bit Streamlined Instruction Processor", Rev. 1, pp. 7-1 through 7-38 (1993).

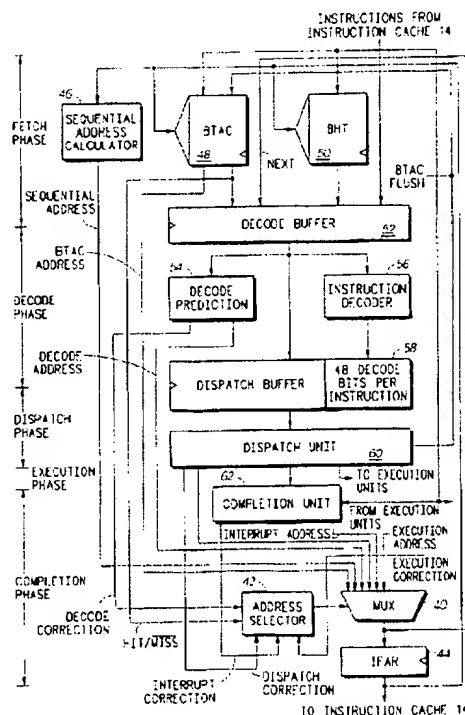
Motorola, "MC88110 Second Generation RISC Microprocessor User's Manual", pp. 8-1 through 8-74 (1991).

Primary Examiner—Eddie P. Chan

Assistant Examiner—Hiep T. Nguyen

**[57] ABSTRACT**

A data processor (10) has a branch target address cache (48) for storing the target addresses of a number of recently taken branch instructions. Normally, each fetch address is compared to the contents of the branch target address cache. If a hit occurs, then the data processor branches to the cached target address. The data processor also has a dispatch unit (60) that invalidates the data stored in the branch target address cache if and when it determines that the branch target address cache "hit" on an instruction that was not a branch instruction at all, a "phantom branch." The data processor thereby automatically invalidates its branch target address cache data after a context switch.

**14 Claims, 4 Drawing Sheets**

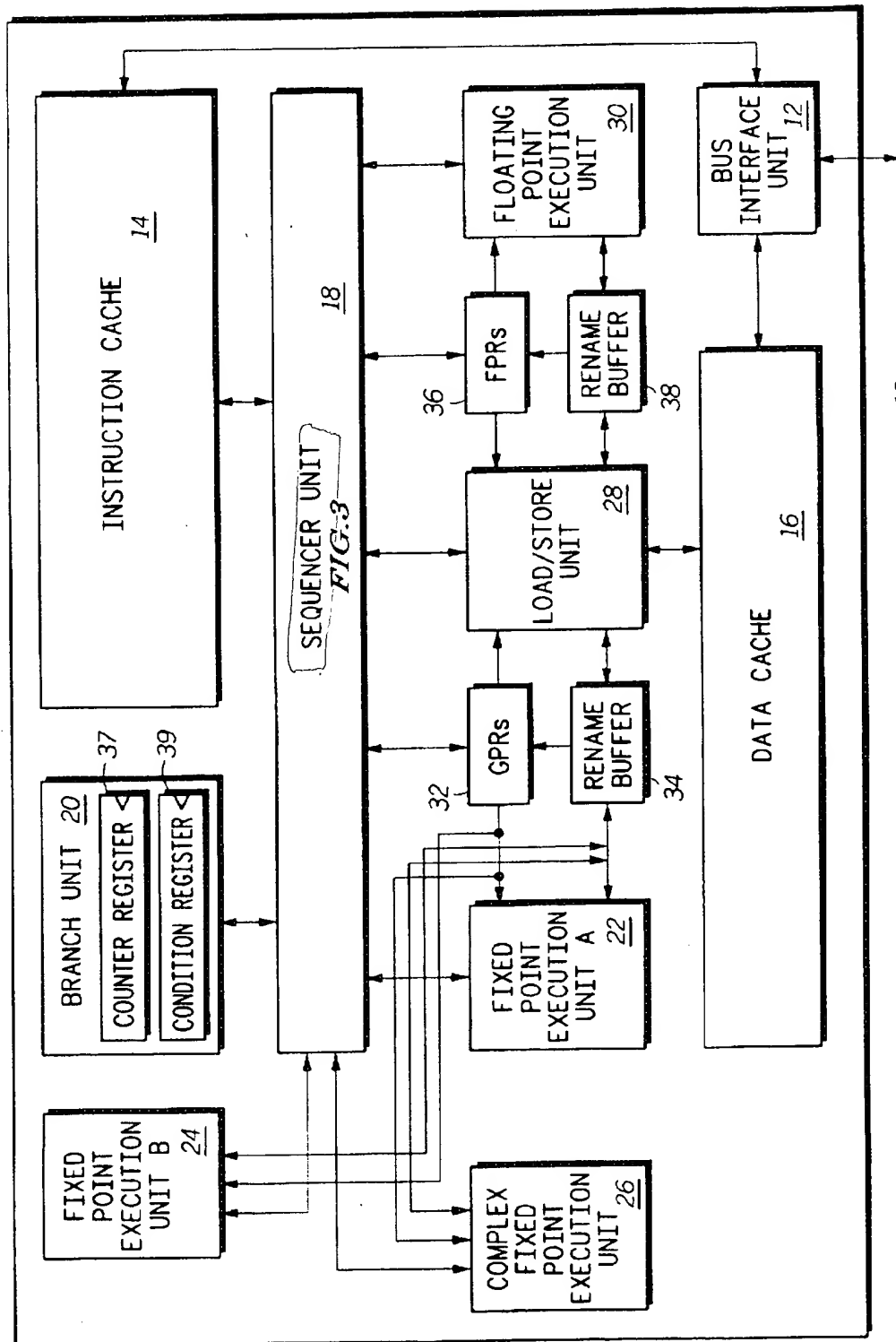
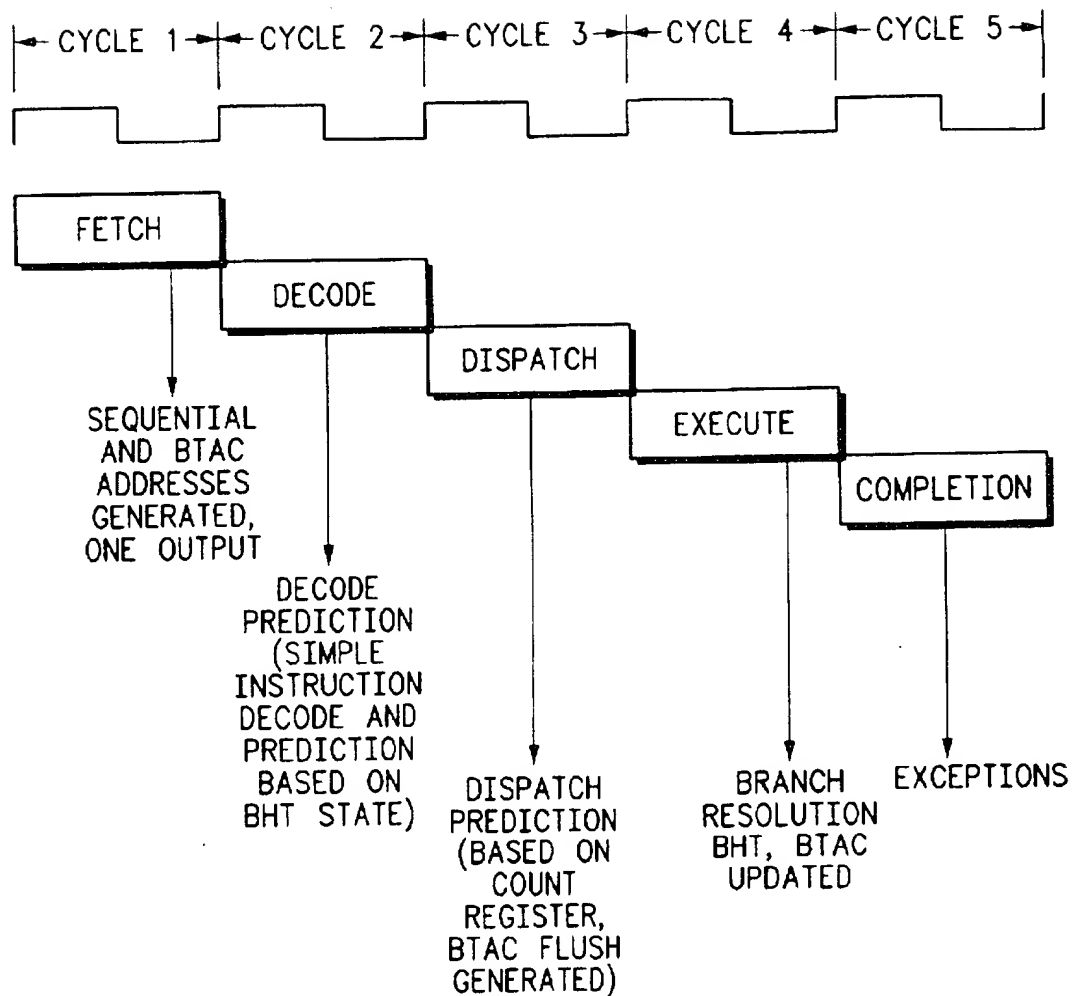
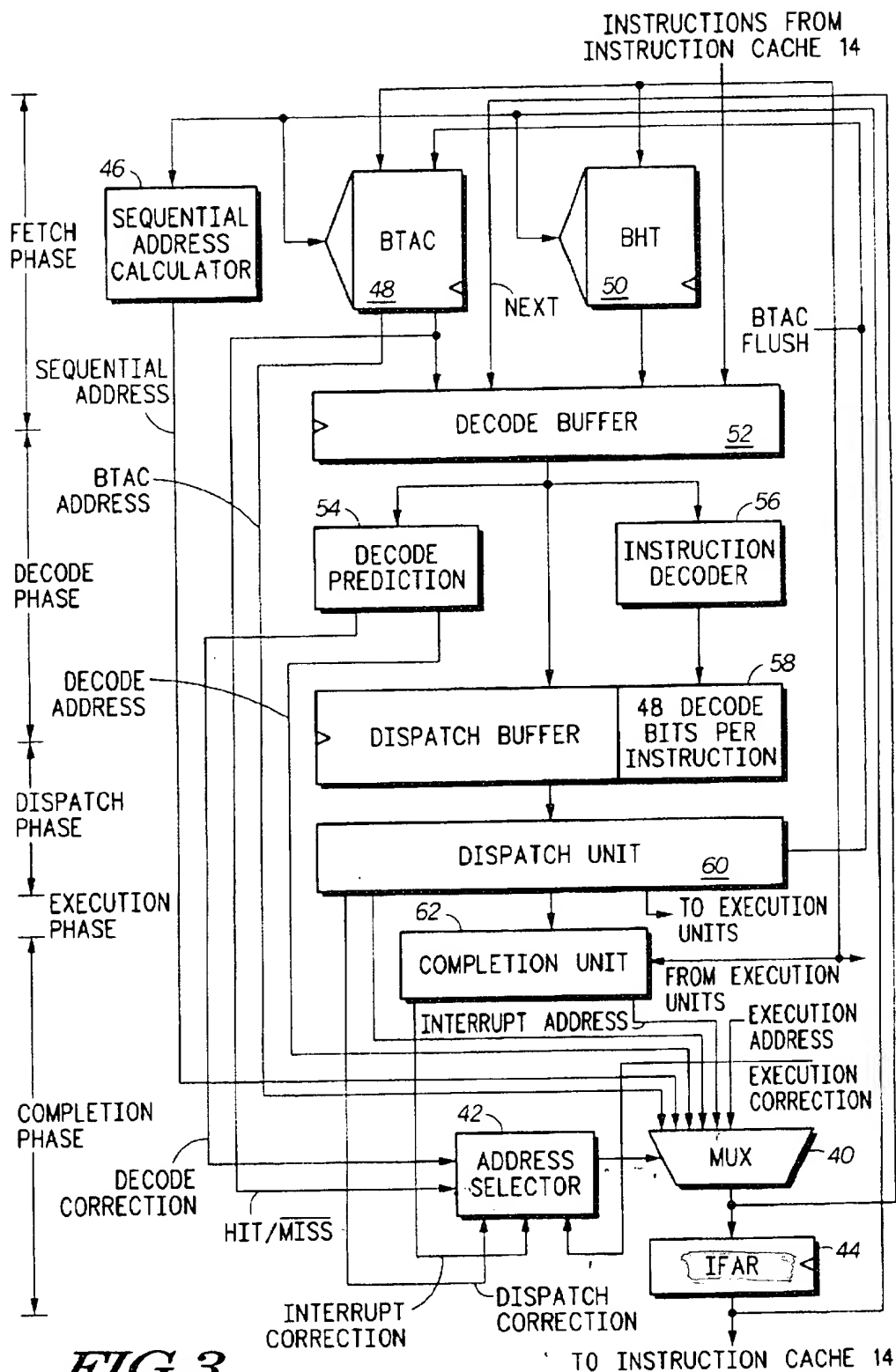
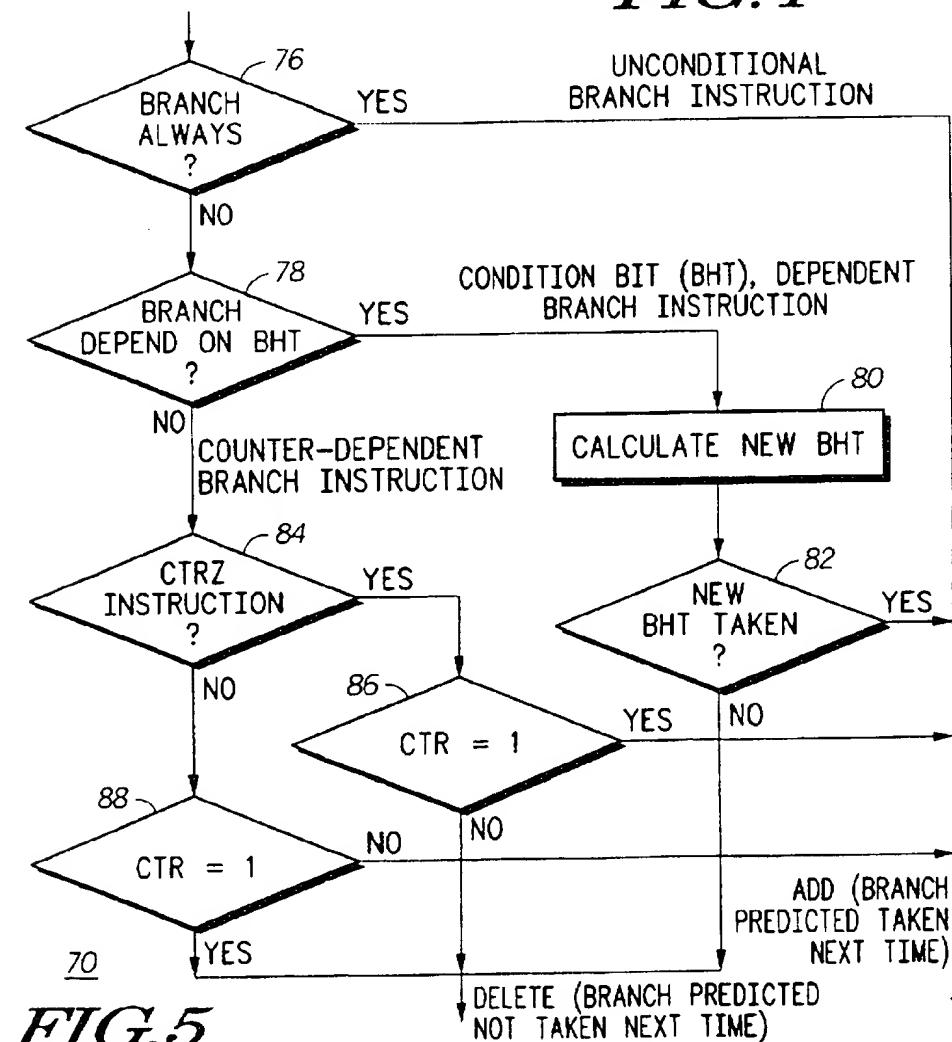
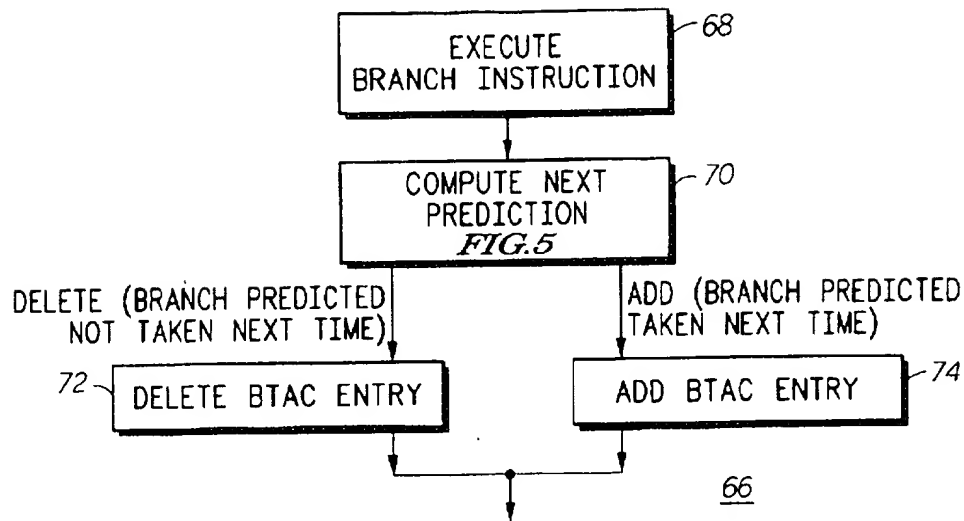


FIG. 1

**FIG.2**





## DATA PROCESSOR WITH BRANCH PREDICTION AND METHOD OF OPERATION

This application is a continuation of prior application Ser. No. 08/192,292, filed on Feb. 4, 1994 entitled DATA PROCESSOR WITH BRANCH PREDICTION AND METHOD OF OPERATION now abandoned.

### RELATED APPLICATION

The present invention is related by subject matter to an U.S. patent application Ser. No. 08/191,888, entitled "Data Processor with Branch Target Address Cache and Method of Operation," to Black et al., filed concurrently herewith.

### FIELD OF THE INVENTION

The present invention generally relates to digital computing systems, and more specifically to a data processor with branch prediction capabilities.

### BACKGROUND OF THE INVENTION

Branch prediction is one technique used to improve data processor performance. If the operands on which a branch instruction depends are not available, then a data processor must either predict the outcome of the branch instruction or must stall the branch instruction until the operands are available. If the data processor stalls, or delays executing the branch instruction, then it can not determine what instructions it needs next. Such a delay will significantly impact the performance of the data processor.

Data processors that use branch prediction techniques make a "guess" each time they receive a branch instruction, act on the guess, and then determine if the guess was correct by executing the instruction. Such a data processor guesses whether a branch will ultimately be taken and "jump" to a new instruction address or whether it will "fall through" to the next sequential instruction. Data processors that predict branch instructions gain performance because they can make an accurate guess faster than they can fully execute the branch instruction. These data processors then need only correct wrong guesses.

Branch target address caches ("BTACs") are devices used to make a branch prediction. BTACs contain addresses to which the data processor has recently branched. These "branch targets" are indexed by the address of the branch instruction which generated them. The data processor will search the BTAC once it determines the address of any instruction that it should next execute. If the address corresponds to a valid entry in the BTAC, then the data processor assumes that it will take the branch again and will automatically branch to the corresponding cached target address. If the address does not correspond to any valid entry in the BTAC, then the data processor will determine the address of its next instruction by some other method.

The addressing schemes used by many data processors complicate the operation of BTACs. Generally, data processors use a first addressing scheme to internally index instructions, virtual addresses, and a second addressing scheme to externally index memory, real addresses. These data processors then use the contents of one or more special purpose arrays to translate addresses from the first addressing scheme to the second addressing scheme. The data processors change the contents of these arrays when, for instance, they begin executing different application programs or subroutines; a context switch. Therefore, the trans-

lation arrays may erroneously map a non-branch instruction to a branch instruction in the BTAC after a context switch if no other action is taken. This mapping will erroneously cause the data processor to fetch instructions at the corresponding cached target address. This scenario is known as a "phantom branch."

One known solution to the phantom branch problem is to include a context switching instruction in the instruction-set architecture of the data processor. Such an instruction causes the data processor to invalidate the entries cached in the BTAC. Unfortunately, this solution, reduces the number of available opcodes in the instruction-set architecture, requires program compiler coordination, and increases the length of the data processor's operating system.

### SUMMARY OF THE INVENTION

In accordance with the present invention, there is disclosed a data processor having a branch prediction unit which substantially eliminates disadvantages of known data processors.

A data processor with branch prediction has a cache array and a dispatch circuitry connected with one another. The cache array stores a plurality of set. Each one of the sets includes a fetch address, a tag and a valid bit. The cache array outputs a fetch address of one of the sets if the corresponding tag is logically equivalent to an input address. The cache array also asserts a control signal when the tag is logically equivalent to the input address. The dispatch circuitry receives the control signal and an instruction indexed by the output fetch address. The dispatch circuitry will set the valid bit of the selected set to a first logic state if the cache array asserts the control signal and the received instruction is not a branch instruction.

A method of operating a data processor comprises the steps of, during a first time, receiving an input address of an instruction in a cache array, outputting a fetch address from the cache array, and asserting a control signal. The method further comprises the steps of, during a second time, receiving the instruction in a dispatch circuitry and setting a valid bit in the cache array to a first logic state if the cache array asserts the control signal and the instruction is not a branch instruction. The output fetch address is a portion of a selected one of a plurality of sets stored in the cache array. Each set contains a fetch address, a valid bit and a tag. The set valid bit corresponds to the stored set of the output fetch address. The tag corresponding to the output fetch address is logically equivalent to a subset of the input address. The cache array asserts the control signal if one of the stored tags matches the subset of the input address.

### BRIEF DESCRIPTION OF THE DRAWINGS

The features and advantages of the present invention will be more clearly understood from the following detailed description taken in conjunction with the accompanying FIGURES where like numerals refer to like and corresponding parts and in which:

FIG. 1 depicts a block diagram of a data processor constructed in accordance with the present invention;

FIG. 2 depicts a timing diagram of an instruction executed by the data processor depicted in FIG. 1;

FIG. 3 depicts a block diagram of the sequencer unit depicted in FIG. 1;

FIG. 4 depicts a flow diagram illustrating how the branch unit depicted in FIG. 1 updates the branch target address cache;

FIG. 5 depicts a flow diagram of one step depicted in FIG. 4.

#### DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

FIG. 1 depicts a block diagram of a data processor 10 constructed in accordance with the present invention. Data processor 10 is a data processor that improves its performance by predicting whether each branch instruction will be taken or will not be taken. Data processor 10 then fetches instructions at the predicted address and begins executing these instructions. Later, data processor 10 resolves whether the branch should have been taken or should not have been taken and performs corrective measures if it predicted incorrectly at the earlier time. According to the disclosed invention, data processor 10 automatically determines whether a context switch has occurred by comparing the output of a branch target address cache ("BTAC") with the decoded instruction. If the BTAC generates a target address for an instruction that is later decoded as a non-branch instruction, then data processor 10 assumes that a context switch must have occurred. In this case, data processor 10 automatically invalidates the contents of the BTAC to ensure that no other instructions erroneously map into the branch instructions of an earlier program. The disclosed invention is automatic, requires no special instruction, and does not increase the length of the data processor's operating system. In addition, the disclosed methodology may be advantageously used to avoid phantom branches in the context of self modifying software code. In these cases, store operations may write non-branch instructions to memory addresses that previously stored branch instructions.

Continuing with FIG. 1, a bus interface unit (hereafter BIU) 12 controls the flow of data between data processor 10 and the remainder of a data processing system (not depicted). BIU 12 is connected to an instruction cache 14 and to a data cache 16. Instruction cache 14 supplies an instruction stream to a sequencer unit 18. Sequencer unit 18 is more fully described below in connection with FIG. 3. Sequencer unit 18 forwards individual instructions to an appropriate execution unit. Data processor 10 has a branch unit 20, a fixed point execution unit A 22, a fixed point execution unit B 24, a complex fixed point execution unit 26, a load/store execution unit 28, and a floating point execution unit 30. Fixed point execution unit A 22, fixed point execution unit B 24, complex fixed point execution unit 26, and load/store execution unit 28 read and write their results to a general purpose architectural register file 32, (labeled GPRs and hereafter GPR file) and to a first rename buffer 34. Floating point execution unit 30 and load/store execution unit 28 read and write their results to a floating point architectural register file 36, (labeled FPRs and hereafter FPR file) and to a second rename buffer 38.

The operation of data processor 10 without the disclosed branch prediction methodology is known in the art. In general, branch unit 20 determines what sequence of programmed instructions is appropriate given the contents of certain data registers and the instructions themselves. Two such registers which branch unit 20 uses to predict branch instructions are a counter register 37 and a condition register 39. Branch unit 20 is described below in connection with FIGS. 4 and 5. Instruction cache 14 provides this sequence of programmed instructions to sequencer 18. If instruction cache 14 does not contain the required instructions, then it will fetch them from a main memory system external to data processor 10 (not shown). (As described below, sequencer unit 18 makes several predictions for each branch instruction before branch unit 20 finally resolves a particular branch instruction.)

Sequencer unit 18 issues the individual instructions of the sequence of programmed instructions to the various execution units 20, 22, 24, 26, 28 and 30. Each of the execution units performs one or more instructions of a particular class of instructions. The particular class of instructions of each execution unit is indicated by the name of the execution unit. For instance, fixed point execution units A and B perform simple mathematical operations on operands expressed in fixed point notation such as addition, subtraction, ANDing, ORing and XORing. Complex fixed point execution 26 performs more complex mathematical operations on operands expressed in fixed point notation such as multiplication and division. Floating point execution unit 30 performs mathematical operations on operands expressed in floating point notation such as multiplication and division.

Fixed point execution units A and B and complex fixed point unit 26 return the results of their operations to designated entries in first rename buffer 34. First rename buffer 34 periodically updates an entry of GPR file 32 with an entry from first rename buffer 34 when all instructions preceding the instruction that generated the result have updated their GPR file entries. Sequencer unit 18 coordinates this updating. Both first rename buffer 34 and GPR file 32 can supply operands to fixed point execution units A and B and to complex fixed point unit 26.

Floating point execution unit 30 returns the results of its operations to designated entries in second rename buffer 38. Second rename buffer 38 periodically updates an entry of FPR file 36 with an entry in second rename buffer 38 when all instructions preceding the instruction that generated the result have updated their FPR file entries. Sequencer unit 18 also coordinates this updating. Both second rename buffer 38 and FPR file 36 supply operands to floating point execution unit 30.

Load/store unit 28 reads data stored in GPR file 32, first rename buffer 34, FPR file 36 or second rename buffer 38 and writes the selected data to data cache 16. This data may also be written to an external memory system (not depicted) depending upon operating characteristics of data processor 10 not relevant to the disclosed invention. Conversely, load/store unit 28 reads data stored in data cache 16 and writes the read data to first rename buffer 34 or second rename buffer 38. If data cache 16 does not contain the required data, then it will fetch it from the main memory system external to data processor 10 via BIU 12.

The operation of data processor 10 with the disclosed branch prediction methodology is described below in connection with FIGS. 2 through 5. In general, data processor 10 is a reduced instruction set computer ("RISC"). Data processor 10 achieves high performance by breaking each instruction into a sequence of smaller steps, each of which may be overlapped in time with steps of other instructions. This performance strategy is known as "pipelining."

FIG. 2 depicts a timing diagram of an instruction executed by data processor 10 depicted in FIG. 1. FIG. 2 also includes various annotations describing the timing of certain branch instruction features. In the depicted embodiment, each instruction is broken into five discrete steps: fetch, decode, dispatch, execute, and completion.

In the fetch stage, the instruction cache 14 retrieves one or more instructions beginning at a memory address identified by sequencer unit 18 or by branch unit 20 in the prior clock cycle. This memory address is called the fetch address. Sequencer unit 18 makes a first branch prediction in the fetch pipeline stage. Sequencer unit 18 uses a BTAC to store a number of pairs of fetch addresses and their corresponding

5

target addresses for a number of recent branch instructions. Sequencer unit 18 will fetch instructions at a particular target address stored in the BTAC if its input fetch address corresponds to the fetch address of the particular target address.

In the decode stage, instructions are decoded by the sequencer unit 18 into a number of control signals for use by subsequent stages. Sequencer unit 18 makes a second branch prediction in the decode pipeline stage. Sequencer unit 18 may generate a fetch address for one or more simple branch instructions. Sequencer unit 18 maintains a history state for each recently encountered branch instruction in a branch history table ("BHT"). The state of each branch instruction determines if sequencer unit 18 will predict the branch instruction taken or not taken.

In the dispatch stage, sequencer unit 18 routes each instruction to the appropriate execution unit after determining that there are no impermissible data or resource dependencies and after reserving a rename buffer entry for the result of the instruction. The dispatch stage is also responsible for supplying operand information for the instructions being dispatched. Sequencer unit 18 makes a third branch prediction in the dispatch pipeline stage. Sequencer unit 18 may generate a fetch address for one or more branch instructions that depend upon the value of counter register 37. It is during the dispatch stage that sequencer unit 18 determines if any phantom branch instructions occurred. If a phantom branch instruction did occur, then sequencer unit 18 invalidates the contents of its BTAC.

In the execute stage, each particular execution unit executes its programmed instruction. Results, if any, are returned either to rename buffer 34 or rename buffer 38, respectively, for integer and floating point results. The contents of counter register 37 and condition register 39 are available at the execute stage for final resolution of all branches. Therefore, the fetch address generated by branch unit 20 in the execute phase is not a prediction.

In the completion stage, sequencer unit 18 updates the architectural register files with the result of a particular instruction stored in a rename buffer after every instruction preceding the particular instruction has so updated the architectural register file. The completion stage also updates all other architectural states in the machine after every instruction preceding the particular instruction has so updated the architectural state.

Generally, each instruction stage takes one machine clock cycle. However, some instructions, such as complex fixed point instructions, require more than one clock cycle to execute. Therefore, there may be a delay between the execution and completion stages of a particular instruction due to the range of times which previous instructions may have taken to execute.

FIG. 3 depicts a block diagram of sequencer unit 18 depicted in FIG. 1. Multiplexer 40 selects a new fetch address each clock cycle from several sources. An address selector 42 selects one of the several sources as the output of multiplexer 40 according to a priority scheme described below. The fetch address is a virtual address that identifies the location of the next instruction or group of instructions that data processor 10 should execute. The fetch address is latched by an instruction fetch address register (labeled and hereafter IFAR) 44 and forwarded to instruction cache 14. In the depicted embodiment, sequencer 18 and branch unit 20 generate a fetch address during each one of the five pipeline stages. It should be noted that each of the (up to) four fetch addresses generated by a particular branch instruction and

6

the return of the exception address will occur at different stages in the pipeline. Conversely, two or more closely spaced branch instructions may cause sequencer 18 and branch unit 20 to forward two different fetch addresses to multiplexer 40 at the same time. In this second case, address selector will choose the fetch address generated in the later pipeline stage.

Sequencer 18 may be conveniently divided into five portions analogous to the five pipeline stages of data processor 10. (The execution portion of sequencer 18 is embodied by the six execution units depicted in FIG. 1.)

In the fetch stage, the virtual address in IFAR 44 is forwarded to a sequential address calculator 46, a BTAC 48, a BHT 50, and a decode buffer 52. The fetch address from IFAR 44 is latched by decode buffer 52 to keep track of the address of instructions in decode buffer 52.

Sequential address calculator 46 generates a fetch address and forwards it to multiplexer 40. "SEQUENTIAL ADDRESS." Sequential address calculator 46 generates the sequential fetch address by adding the number of instructions fetched in the current machine cycle to the contents of IFAR 44.

BTAC 48 may generate a fetch address. "BTAC ADDRESS." and, if so, forwards it to multiplexer 40. BTAC 48 is a fully associative content addressable memory ("CAM") array. BTAC 48 stores a plurality of thirty bit target addresses. Each target address is associated with a twenty-nine bit address tag and a valid bit. BTAC 48 compares the twenty-nine most significant bits of the current fetch address to each stored tag. If the fetch address bits match a stored valid tag, then BTAC 48 outputs the corresponding stored target address to multiplexer 40. In this case, BTAC 48 also asserts a control signal HIT/MISS. BTAC 48 also forwards the predicted target address and the HIT/MISS signal to decode buffer 52 for use later in the pipeline.

Branch unit 20 adds, deletes and updates entries to BTAC 48 during each execution stage. If branch unit 20 indicates that an entry is to be added (branch taken), BTAC 48 is searched. If the entry is already in BTAC 48, then its address information is updated. This is useful because branches that use a register for their branch target information may have incorrect information in BTAC 48. If the entry is not in BTAC 48, then branch unit 20 adds it to BTAC 48, replacing another entry. If the branch unit 20 indicates that an entry is to be deleted (branch not taken), then BTAC 48 is searched and the matching entry (if any) is invalidated. BTAC 48 has two read ports and one write port to facilitate these execution stage operations and the fetch stage operations described above. FIGS. 4 and 5 depict how branch unit 20 updates BTAC 48 and BHT 50.

BHT 50 is a direct mapped random access memory ("RAM") array that maintains a two-bit branch history state for 512 recently encountered branch instructions. Each entry contains two bits to encode four possible states. BHT 50 is indexed with a subset of the fetch address bits from IFAR 44. The latest value of the entry which the current fetch address maps to is read out of BHT 50 and latched in decode buffer 52. Branch unit 20 updates each entry of BHT 50 in the execution stage as described below. BHT 50 has one read port and one write port to facilitate these execution stage operations and the fetch stage operations described above.

Data processor 10 uses a dynamic branch prediction methodology to predict branch instructions which are based on the value of one or more bits in condition register 39. In the PowerPC Architecture, other instructions may modify



the bits of condition register 39 as a prelude to executing a branch instruction. According to a dynamic branch prediction methodology, branch instructions are more likely to be taken or not be taken if previous occurrences of the same branch instructions have been taken or not taken, respectively. Therefore, it is possible to predict whether a branch instruction will be taken or will not be taken by maintaining the history of each branch instruction. The state of a particular branch instruction determines if the branch will be predicted taken or will be predicted not taken during the decode stage of the particular branch instruction. The branch state model for the depicted embodiment is a four-state model: STRONG-NOT-TAKEN, WEAK-NOT-TAKEN, WEAK-TAKEN and STRONG-TAKEN. Sequencer 18 predicts that a branch will not be taken if the instruction's branch state corresponds to STRONG-NOT-TAKEN or WEAK-NOT-TAKEN. Sequencer 18 predicts that a branch will be taken if the instruction's branch state corresponds to STRONG-TAKEN or WEAK-TAKEN. Branch unit 20 updates the state of each entry in BHT 50. The branch unit 20 uses information about the current state of the BHT entry and the direction of an executing branch to calculate the next state of the BHT entry which corresponds to the executing branch instruction. If sequencer 18 incorrectly predicted the branch instruction, then branch unit 20 will update the corresponding entry in BHT 50 from one strong state to the same weak state or from one weak state to the opposite weak state. Conversely, if sequencer unit 18 correctly predicted the branch instruction, then branch unit 20 will update the corresponding entry in BHT 50 from one weak state to the same strong state or from one strong state to the same strong state.

In the decode stage, decode buffer 52 provides instructions, addresses, and control information to a decode prediction unit 54. Decode buffer 52 provides instructions to an instruction decoder 56. A dispatch buffer 58 latches instructions, addresses, and control information from decode buffer 52 and instruction decode information from instruction decoder 56. Decode prediction unit 54 provides a decode stage fetch address to multiplexer 40. "DECODE ADDRESS." Decode prediction unit 54 predicts unconditional branch instructions and predicts branch instructions based on one or more bits in condition register 39. Decode prediction unit 54 predicts unconditional branch instructions by decoding their existence and determining the appropriate fetch address responsive to the branch instruction. Decode prediction unit 54 predicts branch instructions based on one or more bits in condition register 39 by decoding the contents of BHT 50. As described above, the entry in BHT 50 corresponding to the fetch address of the decoded branch instruction determines if the branch instruction is taken or not taken. The branch instruction itself determines the target address.

In the dispatch stage, dispatch buffer 58 supplies instructions and control information to a dispatch unit 60. Dispatch unit 60 does resource allocation and conflict checking for those instructions and sends instructions and control information to the proper execution unit if possible. Dispatch unit 60 also sends control information to a completion unit 62.

Dispatch unit 60 provides a fetch address to multiplexer 40. "DISPATCH ADDRESS." Dispatch unit 60 predicts branch instructions based on an early value of counter register 37. It should be noted that branch unit 20 resolves each branch instruction using the value of counter register 37 at the execution stage. In the PowerPC Architecture, counter register 37 may be conveniently loaded with a value representing the number of iterations of a loop and may be

decremented each iteration to provide a loop exit condition. Dispatch unit 60 predicts whether the branch should be taken or not taken depending upon the particular branch instruction opcode (branch if counter register equals zero, etc.) in view of the value of counter register 37 during the dispatch stage. Branches which use a combination of counter register 37 and condition register 39 are predicted in the dispatch stage. Dispatch unit 60 predicts these branch instructions by assuming that the condition specified by condition register 39 is met and by using the value of counter register 37 during the dispatch stage.

Dispatch unit 60 asserts a BTAC FLUSH control signal to BTAC 48 if it determines that sequencer 18 encountered a phantom branch. Dispatch unit 60 will assert BTAC FLUSH when control information indicates that a particular instruction had a predicted target in BTAC 48 (BTAC 48 asserted HIT/MISS), but the particular instruction is not a branch instruction. Asserting BTAC FLUSH will invalidate all entries in BTAC 48. Dispatch unit 60 then outputs the same sequential address to multiplexer 40 via DISPATCH ADDRESS as output by sequential address calculator 46. Dispatch unit 60 invalidates BTAC 48 because the BTAC entries were likely from a previous context and therefore are no longer useful. Each time a "phantom branch" occurs, sequencer unit 18 wastes several cycles fetching instructions along the phantom path. A BTAC flush prevents this and many other "phantom branches" in the near future.

At the beginning of the execute stage of a branch instruction, branch unit 20 knows the actual value of the branch condition on which the fetch address is based. Typically this condition is some combination of the value of counter register 37 and a bit in condition register 39. At this time final resolution can occur for all branches. Branch unit 20 supplies a fetch address to multiplexer 40 reflecting its calculations. "EXECUTION ADDRESS." The various execution units supply a variety of control information to a completion unit 62 during the execute stage.

In the completion stage, a completion unit 62 updates the architectural machine state. Completion unit 62 provides a fetch address. "INTERRUPT ADDRESS." in the event of an interrupt from some execution unit. Completion unit 62 outputs the INTERRUPT ADDRESS when an instruction that has an interrupt condition is ready to complete. After the interrupt is handled by software, a return from interrupt will typically be executed to resume execution with the instruction following the one which caused the interrupt condition. Completion unit 62 also provides control signals to update architectural state in itself and the execution units. For branch instructions, this step updates the architectural program counter, and counter register 37.

Address selector 42 determines which of up to six addresses it should cause multiplexer 40 to output to IFAR 44. Multiplexer 40 receives the control signal HIT/MISS from BTAC 48, a control signal DECODE CORRECTION from decode prediction unit 54, a control signal DISPATCH CORRECTION from dispatch unit 60, a control signal EXECUTION CORRECTION from branch unit 20 and a control signal INTERRUPT from completion unit 62. The source of each control signal asserts its control signal if the source generates a fetch address different from the fetch address generated during the prior pipeline stage, a corrected fetch address. In general, each stage of the pipeline either resolves a branch or makes a statistically more accurate prediction of the branch direction than the previous stage or stages. Therefore, the later stages in the pipeline have priority over the earlier stages for correction. In the case of the fetch stage, the output of BTAC 48 has priority over the

3rd stage  
Predictive selection

output of the sequential address calculator 46. Each subsequent fetch address correction invalidates all instructions following the corresponding branch instruction in the pipeline. For example if branch unit 20 corrects a prior fetch address, then the decode stage fetch address (of an immediately following branch instruction) is irrelevant. In this case, the decode stage fetch address is based on a branch instruction that should not have been fetched.

FIG. 4 depicts a flow diagram 66 illustrating how branch unit 20 depicted in FIG. 1 updates BTAC 48. According to the PowerPC Architecture, branch unit 20 executes four different classes of branch instructions: branch unconditional, branch conditional based on condition register 39, branch conditional based on a counter register 37, and branch conditional based on a counter register 37 and on condition register 39. Unconditional branch instructions are predicted (resolved) based on their decode during the decode stage. Conditional branch instructions based on condition register 39 are predicted during the decode stage using history bits from BHT 50. Conditional branch instructions based on counter register 37 and conditional branch instructions based on the combination of counter register 37 and condition register 39 are predicted during the dispatch stage. These latter instructions use the decremented value of the counter register 37 and assume any condition register 39 condition is met.

Continuing with FIG. 4, branch unit 20 calculates a new fetch address for each branch instruction it executes, a step 68. Branch unit 20 resolves whether it should continue along the same instruction path (branch not taken) or whether it should jump to a new instruction stream (branch taken) depending upon the branch instruction opcode and the contents of counter register 37 and condition register 39. Branch unit 20 calculates the branch taken fetch address according to the addressing format of the branch instruction.

Branch unit 20 predicts whether it will take each branch or not according to a condition precedent as defined in the opcode of each branch instruction: branch unconditionally, branch if counter register 37 equals zero, branch if counter register 37 does not equal zero, branch if condition register 39 is true, branch if condition register 39 is false, branch if condition register 39 is true and counter register 37 equals zero, branch if condition register 39 is false and counter register 37 equals zero, branch if condition register 39 is true and counter register 37 does not equal zero, or branch if condition register 39 is false and counter register 37 does not equal zero.

Branch unit 20 outputs one of five different addresses for the new fetch address: a sequential address, an absolute target address, an offset target address, the contents of counter register 37, and the contents of a link register (not shown). Branch unit 20 calculates the sequential address by adding the instruction word size (byte, half word, etc.) to the fetch address of the branch instruction. Branch unit 20 calculates the absolute target address by concatenating six zeros, an embedded operand in the branch instruction, and two zeros (most significant bit to least significant bit). Branch unit 20 calculates the offset target address by adding the embedded operand and the branch fetch address. Branch unit 20 selects the sequential address if the branch is resolved as not-taken. Branch unit 20 selects one of the absolute target address, the offset target address, the contents of counter register 37, or the contents of a link register if the branch is resolved as taken. Branch unit 20 selects a particular one of the four possibilities depending upon the addressing format of the branch instruction.

Branch unit 20 then determines how it will predict the same branch instruction the next time it executes the branch

instruction, a step 70. Step 70 is more fully described below in connection with FIG. 5. Branch unit 20 will then either delete the fetch address-target address pair corresponding to the branch instruction from BTAC 48, a step 72, or will add the fetch address-target address pair corresponding to the branch instruction to BTAC 48, a step 74.

FIG. 5 depicts a flow diagram of step 70 depicted in FIG. 4. Branch unit 20 first determines if the branch instruction that it is executing is an unconditional branch instruction, a step 76. If the branch instruction that branch unit 20 is executing is an unconditional branch instruction, then branch unit 20 predicts (actually knows) that it will take the branch instruction the next time it executes the instruction. In this case, branch unit 20 adds the fetch address of the branch instruction and the target address which it calculated in step 68 to BTAC 48. If the branch instruction that branch unit 20 is executing is not an unconditional branch instruction, then branch unit 20 continues to determine what type of branch instruction it has just executed.

Branch unit 20 next determines if the branch instruction that it is executing is a conditional branch instruction, a step 78. As described above, conditional branch instructions depend upon one or more bits in condition register 39. Conditional branch instructions are predicted taken or not taken depending upon the value stored in BHT 50 corresponding to the fetch address of the conditional branch instruction. If the branch instruction that branch unit 20 is executing is a conditional branch instruction, then branch unit 20 calculates a new branch state for BHT 50, a step 80. Branch unit 20 also writes this new branch state to BHT 50 in step 80. Branch unit 20 determines if the new branch state is strong-taken or weak-taken, a step 82. A conditional branch instruction will be predicted taken in both of these cases the next time data processor 10 executes the conditional branch instruction. Therefore, branch unit 20 adds the fetch address of the branch instruction and the target address which it calculated in step 68 to BTAC 48. A conditional branch instruction that has a branch state corresponding to strong-not-taken or weak-not-taken will be predicted not taken the next time data processor 10 executes the conditional branch instruction. Therefore, branch unit 20 deletes the fetch address of the branch instruction and the target address which it calculated in step 68 from BTAC 48.

The branch instruction that branch unit 20 is executing must be based on the value of counter register 37 if the branch instruction is not an unconditional branch instruction nor a conditional branch instruction. As described above, branch unit 20 executes branch instructions that may branch if the value of counter register 37 equals zero or does not equal zero. Branch unit 20 determines if the counter-based branch instruction branches on the condition that the value of counter register 37 equals zero, a step 84.

If the counter register-based branch instruction branches on the condition that the value of counter register 37 is zero, then branch unit 20 determines if the present value of counter register 37 is one, a step 86. Branch unit 20 adds the fetch address of the branch instruction and the target address which it calculated in step 68 to BTAC 48 if the present value of counter register 37 is one. Branch unit 20 deletes the fetch address of the branch instruction and the target address which it calculated in step 68 from BTAC 48 if the present value of counter register 37 is not one. In the PowerPC Architecture, branch unit 20 decrements the value of counter register 37 at the beginning of the execution stage of each branch instruction. Then, branch unit 20 examines the value of counter register 37 as part of branch resolution. Therefore, if the value of counter register 37 is presently one, then the

11

next branch instruction will cause the value of counter register 37 to equal zero. Branch unit 20 assumes that the branch instruction it is executing will also be the next branch instruction it executes. Therefore, the branch-taken requirement that the value of counter register 37 equal zero will be logically true if the present value of the counter is one. Similarly, the branch-taken requirement that the value of counter register 37 equal zero will be logically false if the present value of the counter is not one. (Alternately, branch unit 20 may assume that the value of counter register 37 is not equal to zero (or one) when it predicts whether each branch instruction will be taken or will not be taken the next time it encounters the branch instruction. Statistically, the value of counter register 37 is more often not equal to any particular number than it is equal to the particular number.)

If the counter register-based branch instruction branches on the condition that the value of counter register 37 is not zero, then branch unit 20 determines if the present value of counter register 37 is one, a step 88. Branch unit 20 adds the fetch address of the branch instruction and the target address which it calculated in step 68 to BTAC 48 if the present value of counter register 37 is not one. Branch unit 20 deletes the fetch address of the branch instruction and the target address which it calculated in step 68 from BTAC 48 if the present value of counter register 37 is one. In the PowerPC Architecture, branch unit 20 decrements the value of counter register 37 at the beginning of the execution stage of each branch instruction. Then, branch unit 20 examines the value of counter register 37 as part of branch resolution. Therefore, if the value of counter register 37 is presently one, then the next branch instruction will cause the value of counter register 37 to equal zero. Branch unit 20 assumes that the branch instruction it is executing will also be the next branch instruction it executes. Therefore, the branch-taken requirement that the value of counter register 37 not equal zero will be logically false if the present value of the counter is one. Similarly, the branch-taken requirement that the value of counter register 37 not equal zero will be logically true if the present value of the counter is not one.

It should be noted that the prediction of whether each branch instruction will be taken or not taken the next time branch unit 20 executes it is not solely dependent upon the current resolution of the branch instruction. For instance, a conditional branch instruction could have a branch state equal to strong-taken due to its prior history or due to some start-up state. This branch instruction would be predicted taken during the decode stage of the instruction's pipeline cycle. However, it is possible that the condition on which this instruction is based could cause branch unit 18 to not take the branch during the instruction's execution stage. The branch unit would correct the decode stage fetch address with a execution stage fetch address. Branch unit 20 would also modify the branch state of the branch instruction (step 80) to weak-taken. Branch unit 80 will predict that this conditional branch instruction should be taken the next time it executes the instruction, based on the weak-taken branch state. Therefore, branch unit 18 will add the fetch address-target address pair of this conditional branch instruction to BTAC 48.

Although the present invention has been described with reference to a specific embodiment, further modifications and improvements will occur to those skilled in the art. For instance, the disclosed invention may be incorporated into data processors traditionally classified as complex instruction set computers or CISC machines. Or, the timing of BTAC FLUSH may be generated in a different pipeline stage. Also, certain functional units may be omitted in

12

certain embodiments or relocated to other areas of data processor 10. It is to be understood therefore, that the invention encompasses all such modifications that do not depart from the spirit and scope of the invention as defined in the appended claims.

What is claimed is:

1. A data processor with branch prediction comprising:

a cache array storing a plurality of sets, each one of the plurality of sets comprising a fetch address, a tag and a valid bit, the cache array outputting a fetch address of a selected one of the plurality of sets if the tag of the selected one of the plurality of sets is logically equal to a subset of an input address, the fetch address of the selected one of the plurality of sets being a first fetch address, the cache array asserting a first control signal and the data processor branching to the first fetch address if the tag of the selected one of the plurality of sets is logically equivalent to a subset of the input address;

a decode circuitry receiving an instruction indexed by the received address, the decode circuitry determining whether the instruction is a branch instruction or a non-branch instruction, the decode circuitry asserting a second control signal if the instruction is a non-branch instruction; and

a dispatch circuitry coupled to the cache array and to the decode circuitry, the dispatch circuitry receiving the first control signal and the second control signal, the dispatch circuitry setting the valid bit of the selected one of the plurality of sets to an invalid logic state if the the cache array asserts the first control signal and the decode circuitry asserts the second control signal.

2. The data processor of claim 1 further comprising:

a sequential address calculation circuit generating a second fetch address responsive to the input address; and a multiplexer coupled to the sequential address calculation circuit and to the cache array, the multiplexer circuit outputting the first fetch address responsive to an assertion of the first control signal, the multiplexer circuit outputting the second fetch address responsive to a non-assertion of the first control signal.

3. The data processor of claim 2 wherein the dispatch circuitry further comprises circuitry for setting each valid bit of the plurality of sets to the invalid logic state if the cache array asserts the first control signal and the decode circuitry asserts the second control signal.

4. The data processor of claim 3 wherein the cache array is a content addressable memory.

5. The data processor of claim 1 wherein the dispatch circuitry further comprises circuitry for setting each valid bit of the plurality of sets to the invalid logic state if the cache array asserts the first control signal and the decode circuitry asserts the second control signal.

6. The data processor of claim 5 wherein the cache array is a content addressable memory.

7. The data processor of claim 1 wherein the cache array is a content addressable memory.

8. A method of operating a data processor comprising the steps of:

during a first time period, receiving an input address of a first instruction in a cache array;

outputting a fetch address of a selected one of a plurality of sets if a tag of the selected one of the plurality of sets is logically equal to a subset of the input address, the fetch address of the selected one of the plurality of sets being the first fetch address, each

## 13

one of the plurality of sets comprising a fetch address, a tag and a valid bit;  
 asserting a first control signal by the cache array if the tag of the selected one of the plurality of sets is logically equal to a subset of the input address;  
 branching to a second instruction for execution, said second instruction indexed by said first fetch address;  
 during a second time period, receiving said first instruction in a decode circuitry;  
 decoding whether said first instruction is a branch instruction or is a non-branch instruction;  
 asserting a second control signal by the decode circuitry if said first instruction is a non-branch instruction;  
 during a third time period, by a dispatch circuitry,  
 setting the valid bit of the selected one of the plurality of sets to an invalid logic state if the cache array asserts the first control signal and the decode circuitry asserts the second control signal.  
 9. The method of claim 8 further comprising the steps of:  
 during the first time period, generating a second fetch address in a sequential address calculator circuit responsive to the input address;  
 selecting, by a multiplexer coupled to the cache array and to the sequential address calculator circuit, the first

## 14

fetch address responsive to an assertion of the first control signal, or the second fetch address responsive to a non-assertion of the first control signal.

10. The method of claim 9 wherein the step of setting further comprises the step of setting each valid bit of the plurality of sets to the invalid logic state if the cache array asserts the first control signal and the decode circuitry asserts the second control signal.

11. The method of claim 10 wherein the step of receiving an input address further comprises the step receiving an input address in a content addressable memory.

12. The method of claim 8 wherein the step of setting further comprises the step of setting each valid bit of the plurality of sets to the invalid logic state if the cache array asserts the first control signal and the decode circuitry asserts the second control signal.

13. The method of claim 12 wherein the step of receiving an input address further comprises the step receiving an input address in a content addressable memory.

14. The method of claim 8 wherein the step of receiving an input address further comprises the step receiving an input address in a content addressable memory.

\* \* \* \* \*